Lab 3: “for” loops with incrementing/decrementing index

Taaruni Ananya | Session 008 | CRN: 15623

02/07/2025

Dr. Lan Gao

Introduction

Lab 3: “for” loops with incrementing/decrementing index delves into how a “for” loop works, how to produce “for” loop assembly code with incrementing test expression (which then changes into the decrementing expression), compiling an assembly language file to an executable, running the executable program, and finally an exercise to print even or odd numbers from 1 to 20.

Purpose

The purpose of this lab is to practice compiling a C program to get an assembly language program and look at how a for loop would look in assembly language. It also provides an exercise to log into and access SNOWBALL, along with using sftp to move and access files on the application FileZilla. Additionally, this lab bolstered understanding of loops in C.

Main Objectives

- Learn how a “for” works
- Produce these loops’ assembly language code with incrementing and decrementing test expressions
- Compiling assembly language files to an executable

Brief Explanation

“For” loops in assembly are implemented with a variable to count, comparison, and jump in the first line. Then, goes through each requirement stated in the first line and increments or decrements accordingly, as well as executing any other instructions written in the block.

Methodology/Procedure

The procedure starts with creating a file using ‘vi’ and entering insert mode to write the C code. First starting with starting main(), initializing i, and then the loop header where the i compared, jumped, and incremented. Then the code delves into printing out the resulting

number. After completing the loop's code and closing out of the file, the next step of the procedure is to use gcc to compile and convert C source code into machine executable code.

Documentation

Original incrementing loop (loop.c):

```
// Print numbers from 0 to 5
#include <stdio.h>

int main() {
  int i;

  for (i = 0; i < 6; i++) {
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

```
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc loop.c -o loop
[tananya1@gsuad.gsu.edu@snowball ~]$ ./loop
0 1 2 3 4 5
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc loop.c -S
```

```
.LC0:
    .string "%d "
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $0, -4(%rbp)
    jmp .L2
.L3:
    movl    -4(%rbp), %eax
```

```
.L3:
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    addl    $1, -4(%rbp)
.L2:
    cmpl    $5, -4(%rbp)
    jle .L3
    movl    $10, %edi
    call    putchar
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

Decrementing loop (loop_decrement.c):

```
// Print numbers from 5 to 1
#include <stdio.h>

int main() {
  int i;

  for (i = 5; i > 0; i--) {
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

```
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc loop_decrement.c -o
loop_decrement
[tananya1@gsuad.gsu.edu@snowball ~]$ ./loop_decrement
5 4 3 2 1
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc loop_decrement.c -S
```

```
.LC0:
    .string "%d "
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $5, -4(%rbp)
    jmp .L2
```

```
.L3:
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    subl    $1, -4(%rbp)
.L2:
    cmpl    $0, -4(%rbp)
    jg  .L3
    movl    $10, %edi
    call    putchar
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

Numbers 1 – 20 program (lab3_1to20.c):

```c
#include <stdio.h>

int main() {
    int i;

    for (i = 1; i <= 20; i++) {
        printf("%d ", i);
    }
    printf("\n");
    return 0;
}
```

```
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc lab3_1to20.c -o lab3_1to20
[tananya1@gsuad.gsu.edu@snowball ~]$ ./lab3_1to20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc lab3_1to20.c -S
```

```
.LC0:
    .string "%d "
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $1, -4(%rbp)
    jmp .L2
```

```
.L3:
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    addl    $1, -4(%rbp)
.L2:
    cmpl    $20, -4(%rbp)
    jle .L3
    movl    $10, %edi
    call    putchar
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

Even Numbers program (lab3_1to20even.c):

```
#include <stdio.h>

int main() {
    int i;
    for (i = 2; i <= 20; i += 2) {
        printf("%d ", i);
    }
    printf("\n");
    return 0;
}
```

```
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc lab3_1to20even.c -o lab3_1to20even
[tananya1@gsuad.gsu.edu@snowball ~]$ ./lab3_1to20even
2 4 6 8 10 12 14 16 18 20
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc lab3_1to20even.c -S
```

```
.LC0:
    .string "%d "
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $2, -4(%rbp)
    jmp .L2
```

```
.L3:
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    addl    $2, -4(%rbp)
.L2:
    cmpl    $20, -4(%rbp)
    jle .L3
    movl    $10, %edi
    call    putchar
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

Odd Numbers program (lab3_1to20odd.c):

```c
# include <stdio.h>

int main() {
    int i;
    for (i = 1; i <= 20; i += 2) {
        printf("%d ", i);
    }
    printf("\n");
    return 0;
}
```

```
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc lab3_1to20odd.c -o lab3_1to20odd
[tananya1@gsuad.gsu.edu@snowball ~]$ ./lab3_1to20odd
1 3 5 7 9 11 13 15 17 19
[tananya1@gsuad.gsu.edu@snowball ~]$ gcc lab3_1to20odd.c -S
```

```
.LC0:
    .string "%d "
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $1, -4(%rbp)
    jmp .L2
```

```
.L3:
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    addl    $2, -4(%rbp)
.L2:
    cmpl    $20, -4(%rbp)
    jle .L3
    movl    $10, %edi
    call    putchar
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

Key Differences in Incrementing/Decrementing in Assembly

Incrementing loops in assembly deals with increasing a counter variable during each iteration using code like: **`addl $1, -4(%rbp)`**.Decrementing is decreasing a counter variable during each iteration using code such as: **`subl $1, -4(%rbp)`**. Both programs terminate either when they reach an upper limit (for incrementing) or lower limit (for decrementing).

Questions:

1. What did you change in the program to print numbers from 1 to 20? How difficult was it to figure out what to change? How difficult was it to make the change once you knew what to do, especially in relation to figuring out what to change? (What changes were made to print numbers 1-20? How difficult was it?)

   One of the main changes to print numbers from 1 to 20 was to edit the numbers in the loop control statement. It wasn't too hard since I've studied and programmed in Java before, the syntax is very similar. The main change is shown in the pictures below.

   Before:

```
for (i = 0; i < 6; i++)
```

   After:

```
for (i = 1; i <= 20; i++)
```

2. If the loop variable is an integer, and you divide it by 2, is the result also an integer? Why or why not?

   Yes, the result if a loop variable is an integer and divided by 2 would also be an integer. This is because the new number is stored in the **int** variable of i.

3. Once you had it working for odd numbers, how easy was it to print the even numbers? (Use relative terms like "much more difficult", "somewhat more difficult", "about the same", "somewhat easier, and "much easier". Then state why.

   As I stated above, I've worked with Java before and studied the code for loops in depth during this lab, figuring out both the C and assembly wasn't too difficult. I only struggled in the beginning when I was trying to remember the syntax but it is about the same.

Key Assembly Code Observations

```
Some of the main differences are:


- Filename: from loop.c to loop_decrement.c

- Value: 0 in loop.c to 5 in loop_decrement.c

- Increment to decrement

- The comparision condition changed from <= 5 to > 0
```

```
[tananya1@gsuad.gsu.edu@snowball ~]$ diff loop.s loop_decrement.s
1c1
<   .file   "loop.c"
---
>   .file   "loop_decrement.c"
17c17
<   movl    $0, -4(%rbp)
---
>   movl    $5, -4(%rbp)
25c25
<   addl    $1, -4(%rbp)
---
>   subl    $1, -4(%rbp)
27,28c27,28
<   cmpl    $5, -4(%rbp)
<   jle .L3
---
>   cmpl    $0, -4(%rbp)
>   jg  .L3
```

- movl instruction deals with transferring data between registers or memory locations.
- addl instruction deals with adding values to registers or memory locations.
- subl instruction deals with subtracting values to registers or memory locations.
- cmpl instruction deals with comparing two values without storing the subtraction's result.
- jle instructions (jump if less than or equal) instruction deals with transferring execution to a specified label.
- jg (jiump if greater) instruction deals with moving execution to a specific label if a comparison assigned to the case is more than zero.

Conclusion

During this lab, I learned how to produce "for" loop assembly code with incrementing test expression (which then changes into the decrementing expression), compiling an assembly language file to an executable, running the executable program, and finally an exercise to print even or odd numbers from 1 to 20. Some of the challenges were navigating the complex and new symbols and code in assembly. While it was easy to work with the C code with my past experience of Java, the assembly code was a new experience and something I'm still getting used to after four weeks in Computer Organization and Programming. I gained a new understanding of instructions like addl, subl, movl, cmpl, jg, and jle. This lab gave me hands-on experience, bolstering my understanding.